1. **Basic Operations**

1. **Familiarize the students with the basic functions of OpenCV such as cv2.imread(), cv2.imshow(), and cv2.waitKey() and Write a basic Python program to load and display an image using OpenCV**

c

* **cv2.imread():** This function is used to read an image from a file into a numpy array, which can then be manipulated using OpenCV. The function takes one argument, which is the path to the image file. It returns a numpy array representing the image.
* **cv2.imshow():** This function is used to display an image in a window on the screen. The function takes two arguments: the name of the window (as a string), and the image data (as a numpy array).
* **cv2.waitKey():** This function waits for a key event from the user. It takes one argument, which is the delay in milliseconds. If the delay is set to 0, the function waits indefinitely for a key event.

**Procedure:**

**Step 1:** Load the image from the file

**Step 2:** Display the image in a window

**Step 3:** Wait for a key press and then close the window

**Program:**

import cv2

# Load the image from the file

img = cv2.imread('image.jpg')

# Display the image in a window

cv2.imshow('Image', img)

# Wait for a key press and then close the window

cv2.waitKey(0)

cv2.destroyAllWindows()

1. **Write a program to read a video file and display it on the screen**

**Procedure:**

**Step 1:** Import the necessary libraries: cv2 for OpenCV and numpy for numerical operations.

**Step 2:** Define the video file path using cv2.VideoCapture() function.

**Step 3:** Check if the video file is opened correctly by calling cap.isOpened() function.

**Step 4:** Loop through each frame of the video using while loop.

**Step 5:** Read the frame using cap.read() function.

**Step 6:** Check if the frame is empty using if statement.

**Step 7:** Display the frame on the screen using cv2.imshow() function.

**Step 8:** Wait for a key event to occur using cv2.waitKey() function.

**Step 9:** Check if the key pressed is 'q' to quit the video using if statement.

**Step 10:** Release the video capture object using cap.release() function.

**Step 11:** Close all windows using cv2.destroyAllWindows() function.

**2. Intensity Transformations**

**a) Write a program to perform Image Negation**

**Procedure:**

**Step 1:** Import the necessary OpenCV libraries.

**Step 2:** Load the input image using the cv2.imread() function.

**Step 3:** Convert the input image to grayscale using cv2.cvtColor() function.

**Step 4:** Compute the negative of the grayscale image using the cv2.bitwise\_not() function.

**Step 5:** Display the input image and the negative image using the cv2.imshow() function.

**Step 6:** Wait for a key press using the cv2.waitKey() function.

**Step 7:** If the key pressed is 's', save the negative image using the cv2.imwrite() function.

**Step 8:** Destroy all windows using the cv2.destroyAllWindows() function.

**Program:**

import cv2

# Load input image

img = cv2.imread('input\_image.jpg')

# Convert input image to grayscale

gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

# Compute negative of the grayscale image

neg = cv2.bitwise\_not(gray)

# Display the input image and negative image

cv2.imshow('Input Image', img)

cv2.imshow('Negative Image', neg)

# Wait for a key press

key = cv2.waitKey(0)

# If the key pressed is 's', save the negative image

if key == ord('s'):

Cn mv2.imwrite('negative\_image.jpg', neg)

# Destroy all windows

cv2.destroyAllWindows()

**b) Write a program to perform Log transformation on the image**

**Procedure: w**

**Step 1:** Import the necessary OpenCV modules and the input image.

**Step 2:** Define the gamma value and the constant C for log transformation.

**Step 3:** Convert the input image to grayscale if it is a color image.

**Step 4:** Perform the log transformation using the cv2.log() function and the given formula:

output\_image = C \* log(1 + input\_image)

**Step 6:** Display the input and output images side by side using cv2.hconcat() and cv2.imshow() functions.

**Step 7:** Save the output image using the cv2.imwrite() function.

**Program:**

import cv2

import numpy as np

# Read input image

img = cv2.imread('input\_image.jpg')

# Define gamma value for gamma correction

gamma = 1.5

# Define constant C for log transformation

C = 20

# Convert input image to grayscale

gray\_img = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

# Perform log transformation

log\_img = C \* np.log(1 + gray\_img)

# Display input and output images side by side

result = cv2.hconcat([gray\_img, log\_img])

cv2.imshow(Log transformation', result)

# Save the output image

cv2.imwrite('output\_image.jpg', result)

cv2.waitKey(0)

cv2.destroyAllWindows()

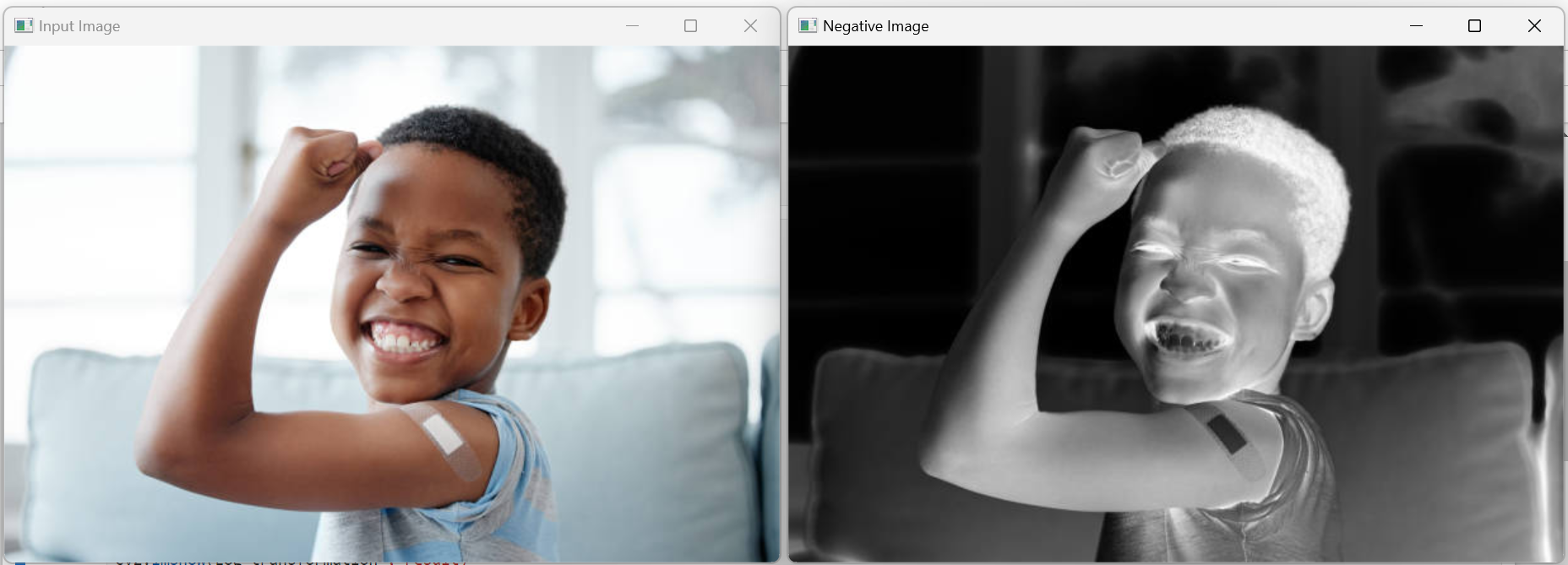


A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated



**Summary:**

* Load an image from a file.
* Convert the image to grayscale.
* Create the negative of the grayscale image.
* Display both the original and negative images.
* Wait for a key press. If 's' is pressed, save the negative image.
* Close all image display windows.

1. **Image Enhancement**
2. **Write a program to enhance the image using piece-wise linear transformation by gray-level slicing.**

import cv2

import numpy as np # You forgot to import numpy

# Read the image in grayscale

img = cv2.imread('background.jpg', 0)

# Find width and height of the image

row, column = img.shape

# Create a zeros array to store the sliced image

img1 = np.zeros((row, column), dtype='uint8')

# Specify the min and max range

min\_range = 80

max\_range = 140

# Loop over the input image and if pixel value lies in the desired range set it to 255

# otherwise set it to the previous pixel value (ensuring i-1, j-1 are valid indices)

for i in range(row):

for j in range(column):

if img[i, j] > min\_range and img[i, j] < max\_range:

img1[i, j] = 255

else:

if i > 0 and j > 0:

img1[i, j] = img[i-1, j-1]

else:

img1[i, j] = img[i, j] # For boundary pixels, use the same pixel value

# Save the original and sliced images

cv2.imwrite('Original.jpg', img)

cv2.imwrite('slicedimage.jpg', img1)

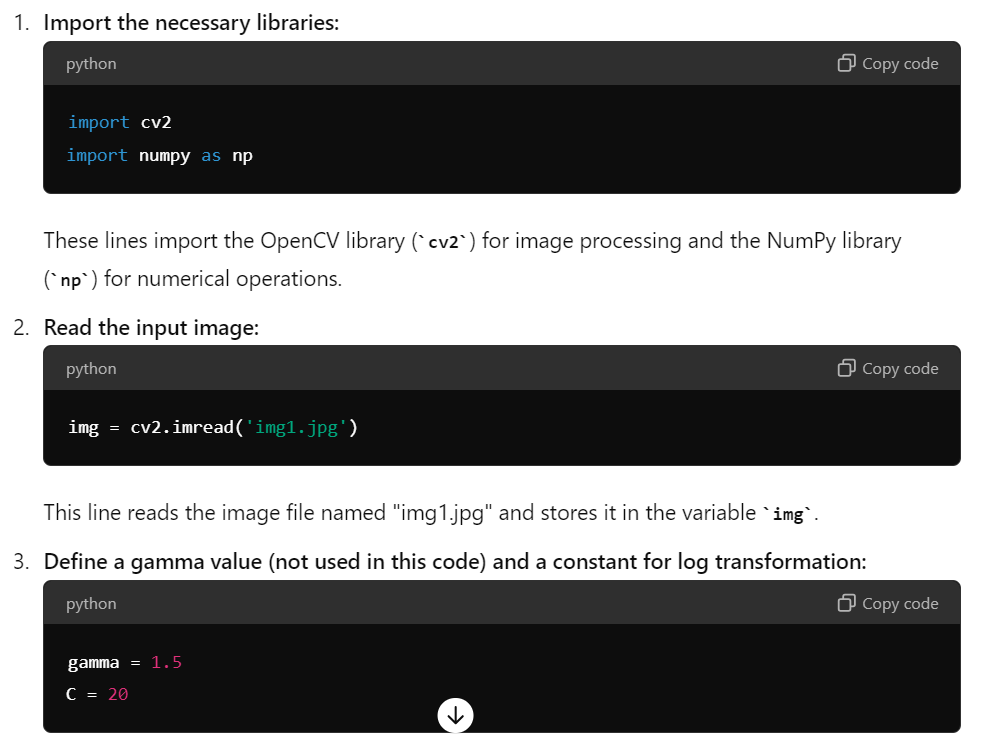
# Display the results (optional)

cv2.imshow('Original Image', img)

cv2.imshow('Sliced Image', img1)

cv2.waitKey(0)

cv2.destroyAllWindows()



A screenshot of a computer program

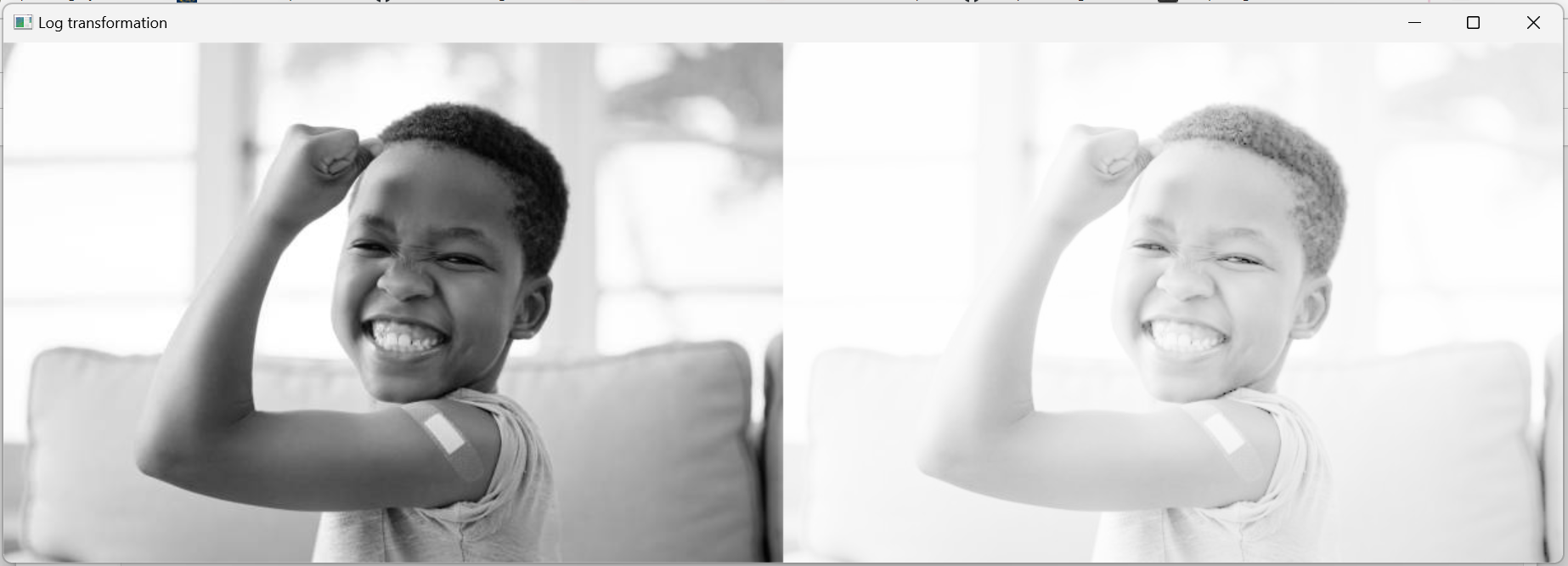
Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

****

**Summary:**

* Import necessary libraries.
* Read an input image.
* Define constants for image transformation.
* Convert the image to grayscale.
* Apply a log transformation to enhance darker regions.
* Normalize the transformed image to the range [0, 255].
* Convert the normalized image to uint8 type.
* Display the original and transformed images side by side.
* Save the concatenated image.
* Wait for a key press and close the display windows.

**3.Histogram Equalization**

**b) Write a program to enhance the contrast of the image using Histogram Equalization**

# Importing OpenCV

import cv2

# Importing numpy

import numpy as np

# Importing matplotlib.pyplot

import matplotlib.pyplot as plt

# Reading the image

img = cv2.imread(r"C:\Users\tushi\Downloads\PythonGeeks\deer.png")

img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB) # Convert BGR to RGB for correct color display

img\_gray = cv2.cvtColor(img, cv2.COLOR\_RGB2GRAY) # Convert to grayscale for processing

# Setting the grid size for multiple plots

plt.figure(figsize=(20, 20))

# Plotting the original image

plt.subplot(221)

plt.title('Original')

plt.imshow(img)

plt.axis('off') # Hide axes for better visualization

# Plotting the histogram for the grayscale image

img\_hist = cv2.calcHist([img\_gray], [0], None, [256], [0, 256])

plt.subplot(222)

plt.title('Grayscale Histogram')

plt.plot(img\_hist)

# Plotting the histogram using the ravel function

plt.subplot(223)

plt.hist(img\_gray.ravel(), 256, [0, 256])

plt.title('Histogram using ravel()')

# Applying Histogram Equalization to the grayscale image

equalized\_img = cv2.equalizeHist(img\_gray)

# Plotting the equalized image

plt.subplot(224)

plt.title('Equalized Image')

plt.imshow(equalized\_img, cmap='gray')

plt.axis('off')

# Display the plots

plt.show()

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

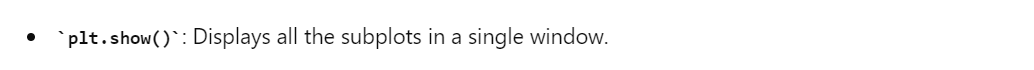
Description automatically generated

A screenshot of a computer program

Description automatically generated

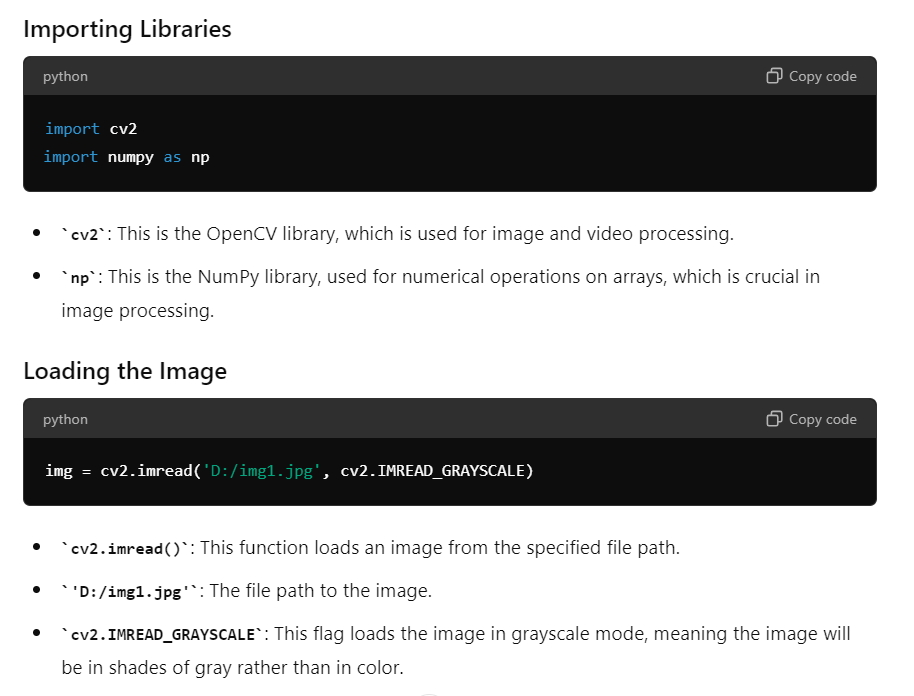
A screenshot of a computer program

Description automatically generated



**4. Image Denoising Using Filters**

**Implement the following low-pass and high-pass filters to improve the quality of the image: a) mean filter b) median filter c) Laplacian filter**



A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A collage of a couple of people

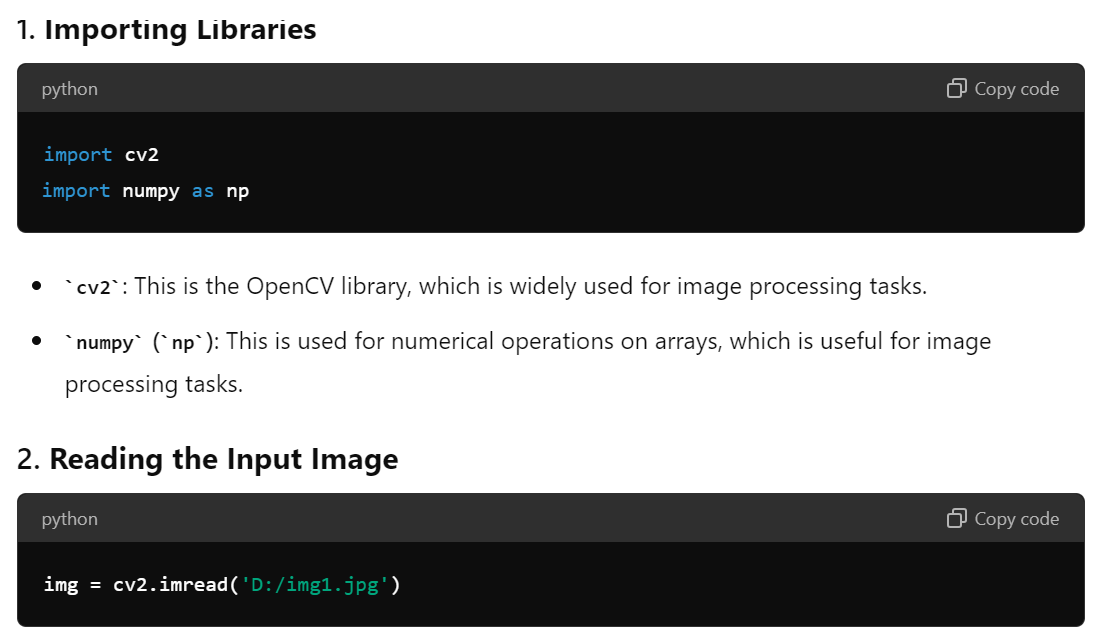
Description automatically generated

A child smiling and a child sitting on a couch

Description automatically generated

**5. Edge Detection**

**Implement edge detection using Sobel and Canny edge detectors using OpenCV**



A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

A black rectangular object with text

Description automatically generated

A child smiling and a black and white image

Description automatically generated

#canny edge detector

import cv2

import numpy as np

# Read the input image

img = cv2.imread('D:/img1.jpg')

# Check if the image is loaded successfully

if img is None:

print("Error: Image not loaded.")

exit()

# Convert the image to grayscale

gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

# Apply Gaussian blur to remove noise

blur = cv2.GaussianBlur(gray, (5, 5), 0)

# Apply Sobel operator to detect edges in x and y directions

sobelx = cv2.Sobel(blur, cv2.CV\_64F, 1, 0, ksize=3)

sobely = cv2.Sobel(blur, cv2.CV\_64F, 0, 1, ksize=3)

# Combine the edges detected in x and y directions using magnitude

edges = np.sqrt(sobelx\*\*2 + sobely\*\*2)

# Convert the magnitude to an 8-bit image

edges = np.uint8(np.absolute(edges))

# Optionally, apply a binary threshold to highlight edges

\_, edges = cv2.threshold(edges, 50, 255, cv2.THRESH\_BINARY)

# Display the original image and the edge-detected image

cv2.imshow('Original Image', img)

cv2.imshow('Edge-Detected Image', edges)

# Save the edge-detected image

cv2.imwrite('output.jpg', edges)

# Wait for user to close the window

cv2.waitKey(0)

# Close all windows

cv2.destroyAllWindows()

**6. Feature Description Using SIFT**

**Write a program to Match two images based on features extracted by SIFT algorithm**

#SIFT

# Import necessary libraries

import cv2

import matplotlib.pyplot as plt

import numpy as np

# Load the image from the specified path

image1 = cv2.imread('D:/subjects/Principles of Computer Vision/CV/CV/CV lab/img.png')

# Check if the image is loaded correctly

if image1 is None:

print("Error: Image not found.")

else:

# Convert the training image to RGB

training\_image = cv2.cvtColor(image1, cv2.COLOR\_BGR2RGB)

# Convert the training image to grayscale

training\_gray = cv2.cvtColor(training\_image, cv2.COLOR\_RGB2GRAY)

# Create test image by adding Scale Invariance and Rotational Invariance

test\_image = cv2.pyrDown(training\_image)

test\_image = cv2.pyrDown(test\_image)

num\_rows, num\_cols = test\_image.shape[:2]

rotation\_matrix = cv2.getRotationMatrix2D((num\_cols/2, num\_rows/2), 30, 1)

test\_image = cv2.warpAffine(test\_image, rotation\_matrix, (num\_cols, num\_rows))

test\_gray = cv2.cvtColor(test\_image, cv2.COLOR\_RGB2GRAY)

# Display training image and testing image

fig, plots = plt.subplots(1, 2, figsize=(20, 10))

plots[0].set\_title("Training Image")

plots[0].imshow(training\_image)

plots[0].axis('off') # Hide axes for better visualization

plots[1].set\_title("Testing Image")

plots[1].imshow(test\_image)

plots[1].axis('off') # Hide axes for better visualization

plt.show()



# SIFT feature detection

sift = cv2.SIFT\_create()

# Detect keypoints and descriptors for both images

keypoints\_train, descriptors\_train = sift.detectAndCompute(training\_gray, None)

keypoints\_test, descriptors\_test = sift.detectAndCompute(test\_gray, None)

# Draw the keypoints on the images

training\_image\_with\_keypoints = cv2.drawKeypoints(training\_image, keypoints\_train, None)

test\_image\_with\_keypoints = cv2.drawKeypoints(test\_image, keypoints\_test, None)

# Display the images with keypoints

fig, plots = plt.subplots(1, 2, figsize=(20, 10))

plots[0].set\_title("Training Image with Keypoints")

plots[0].imshow(training\_image\_with\_keypoints)

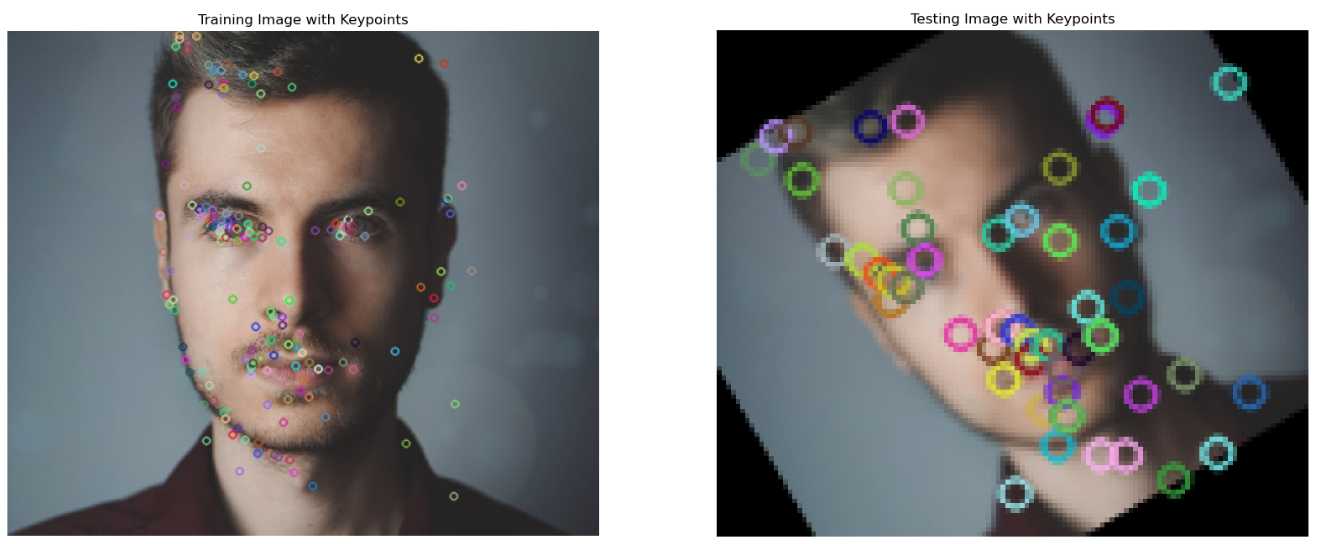
plots[0].axis('off')

plots[1].set\_title("Testing Image with Keypoints")

plots[1].imshow(test\_image\_with\_keypoints)

plots[1].axis('off')

plt.show()



**7.** **Geometric Transformations**

**Implement the following geometric transformations on the image using OpenCV: a)Rotation b)Scaling c) Translation d)Shear**

**Procedure:**

**Step 1:** Read the input image using cv2.imread() function.

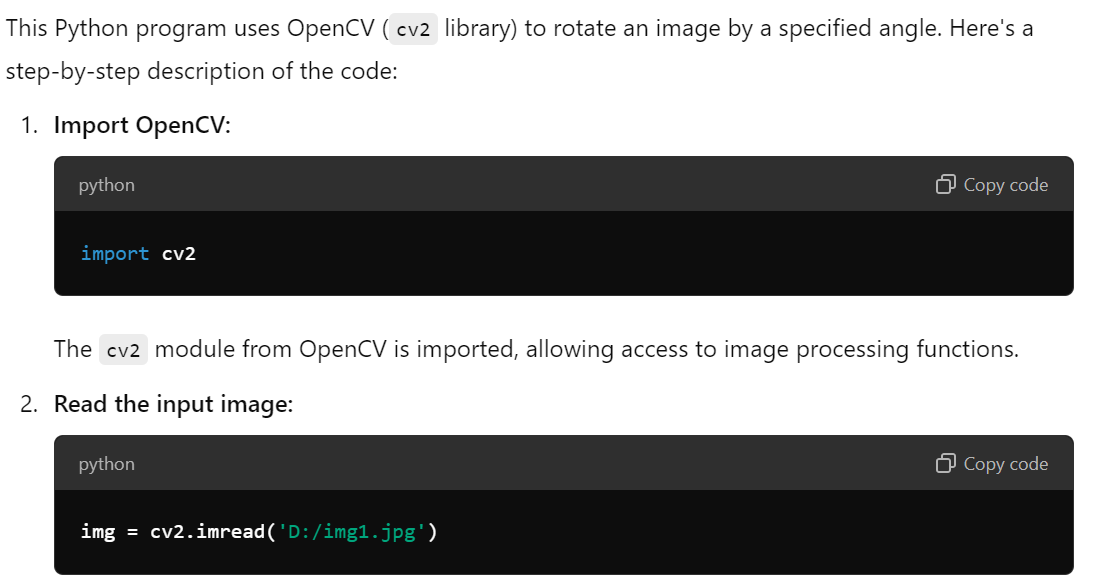
**Step 2:** Define the transformation matrix M to perform the desired transformation using cv2.getRotationMatrix2D(), cv2.getAffineTransform(), or cv2.getPerspectiveTransform() functions depending on the type of transformation required.

**Step 3:** Apply the transformation to the image using cv2.warpAffine() or cv2.warpPerspective() function depending on the type of transformation defined in step 2.

**Step 4:** Display the transformed image using cv2.imshow() function.

**Step 5:** Wait for a key event using cv2.waitKey() function.

**Step 6:** Release the memory and destroy all windows using cv2.destroyAllWindows() function.



A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A black rectangular object with text

Description automatically generated

**Program:**

**# program to rotate the image**

import cv2

# Read the input image

img = cv2.imread(' D:/img1.jpg')

# Define the rotation angle

angle = 45

# Get the image dimensions

h, w = img.shape[:2]

# Calculate the rotation matrix M

M = cv2.getRotationMatrix2D((w/2, h/2), angle, 1)

# Apply the rotation to the image

rotated\_img = cv2.warpAffine(img, M, (w, h))

# Display the original and rotated images

cv2.imshow('Original Image', img)

cv2.imshow('Rotated Image', rotated\_img)

# Wait for a key event

cv2.waitKey(0)

# Release the memory and destroy all windows

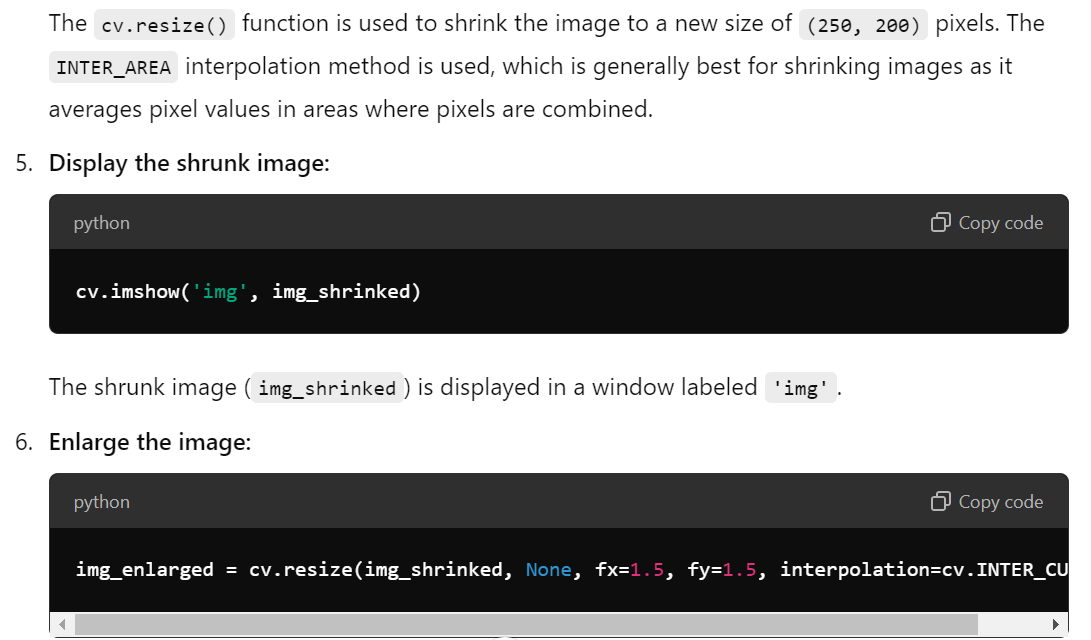
cv2.destroyAllWindows()

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated



A screenshot of a computer

Description automatically generated

A black rectangular object with text

Description automatically generated

#Program to scale an image:

**import** numpy as np

**import** cv2 as cv

img **=** cv.imread(' D:/img1.jpg', 0)

rows, cols **=** img.shape

img\_shrinked **=** cv.resize(img, (250, 200),

                 interpolation**=**cv.INTER\_AREA)

cv.imshow('img', img\_shrinked)

img\_enlarged **=** cv.resize(img\_shrinked, None,

                         fx**=**1.5, fy**=**1.5,

                         interpolation**=**cv.INTER\_CUBIC)

cv.imshow('img', img\_enlarged)

cv.waitKey(0)

cv.destroyAllWindows()



A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

A black rectangular object with white text

Description automatically generated

**#program to translate an image**

**import** numpy as np

**import** cv2 as cv

img **=** cv.imread(' D:/img1.jpg', 0)

rows, cols **=** img.shape

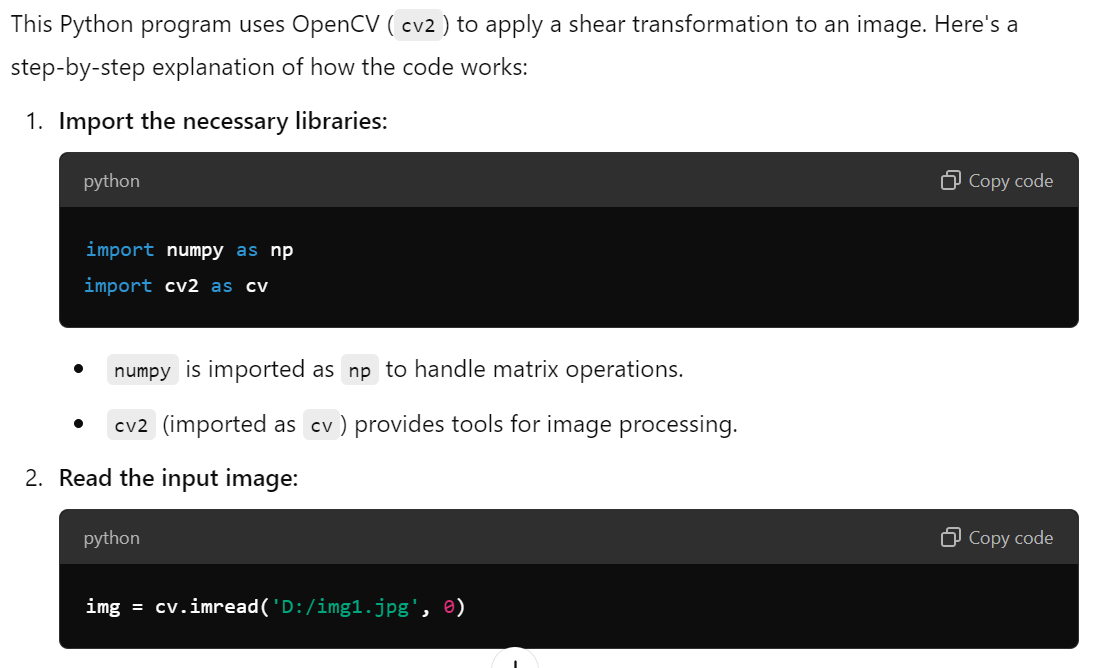
M **=** np.float32([[1, 0, 100], [0, 1, 50]])

dst **=** cv.warpAffine(img, M, (cols, rows))

cv.imshow('img', dst)

cv.waitKey(0)

cv.destroyAllWindows()



A screenshot of a computer

Description automatically generated

A screenshot of a computer

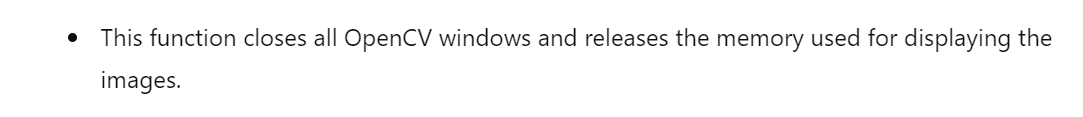
Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated



**#program to shear an image**

import numpy as np

import cv2 as cv

img = cv.imread(' D:/img1.jpg', 0)

rows, cols = img.shape

M = np.float32([[1, 0.5, 0], [0, 1, 0], [0, 0, 1]])

sheared\_img = cv.warpPerspective(img, M, (int(cols\*1.5), int(rows\*1.5)))

cv.imshow('img', sheared\_img)

cv.waitKey(0)

cv.destroyAllWindows()

**8. Tracking of Moving Objects**

**Implement optical flow and track moving objects in a video using OpenCV**

**Procedure:**

**Step 1:** Load the input video and read the first frame.

**Step 2:** Convert the first frame to grayscale and use it as the reference image.

**Step 3:** Loop over the remaining frames in the video:

a. Convert the current frame to grayscale.

b. Compute the optical flow using a method such as Lucas-Kanade or Farneback.

c. Compute the magnitude and direction of the optical flow vectors.

d. Threshold the magnitude to remove noise and small motions.

e. Compute the contours of the thresholded motion map.

f. Filter the contours by area and aspect ratio to remove noise and false detections.

g. Draw bounding boxes around the remaining contours and track them using a simple Kalman filter or a more sophisticated tracker such as the SORT algorithm.

h. Update the reference image to the current frame.

i. Display the output video with the bounding boxes around the moving objects.

import cv2

import numpy as np

# Load the input video

cap = cv2.VideoCapture('D:\small\_video\_10second.mp4')

# Read the first frame and convert it to grayscale

ret, first\_frame = cap.read()

if not ret:

print("Error: Couldn't read the video.")

exit()

prev\_gray = cv2.cvtColor(first\_frame, cv2.COLOR\_BGR2GRAY)

# Create a mask for drawing the optical flow vectors

mask = np.zeros\_like(first\_frame)

mask[..., 1] = 255 # Set saturation to maximum for colorful display

while cap.isOpened():

# Capture the next frame

ret, frame = cap.read()

if not ret:

break

# Convert the current frame to grayscale

gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

# Calculate the optical flow between the previous and current frame

flow = cv2.calcOpticalFlowFarneback(prev\_gray, gray, None, 0.5, 3, 15, 3, 5, 1.2, 0)

# Calculate the magnitude and angle of the flow vectors

magnitude, angle = cv2.cartToPolar(flow[..., 0], flow[..., 1])

# Set the mask’s hue to the angle of the flow

mask[..., 0] = angle \* 180 / np.pi / 2

# Normalize the magnitude to the range 0 to 255

mask[..., 2] = cv2.normalize(magnitude, None, 0, 255, cv2.NORM\_MINMAX)

# Convert the mask to a BGR image

rgb\_flow = cv2.cvtColor(mask, cv2.COLOR\_HSV2BGR)

# Overlay the flow on the original frame

output = cv2.addWeighted(frame, 1, rgb\_flow, 2, 0)

# Display the result

cv2.imshow('Optical Flow', output)

# Break the loop if the user presses the 'q' key

if cv2.waitKey(1) & 0xFF == ord('q'):

break

# Update the previous frame to the current frame

prev\_gray = gray.copy()

# Release the video capture and close windows

cap.release()

cv2.destroyAllWindows()